Numerische Mathematik (2023) 155:345–376
https://doi.org/10.1007/s00211-023-01377-5

Numerische Mathematik

Numerical stability and tensor nuclear norm

Zhen Dai[1] · Lek-Heng Lim[1]

Received: 19 July 2022 / Revised: 4 June 2023 / Accepted: 11 October 2023 /
Published online: 3 November 2023

Abstract
We present a notion of bilinear stability, which is to numerical stability what bilinear complexity is to time complexity. In bilinear complexity, an algorithm for evaluating a bilinear operator $\beta : \mathbb{U} \times \mathbb{V} \to \mathbb{W}$ is a decomposition $\beta = \varphi_1 \otimes \psi_1 \otimes w_1 + \cdots + \varphi_r \otimes \psi_r \otimes w_r$; the number of terms r captures the speed of the algorithm; and its smallest possible value, i.e., the tensor rank of β, quantifies the speed of a fastest algorithm. Bilinear stability introduces norms to the mix: The growth factor of the algorithm $\|\varphi_1\|_* \|\psi_1\|_* \|w_1\| + \cdots + \|\varphi_r\|_* \|\psi_r\|_* \|w_r\|$ captures the accuracy of the algorithm; and its smallest possible value, i.e., the tensor nuclear norm of β, quantifies the accuracy of a stablest algorithm. To substantiate this notion, we establish a bound for the forward error in terms of the growth factor and present numerical evidence comparing various fast algorithms for matrix and complex multiplications, showing that larger growth factors correlate with less accurate results. Compared to similar studies of numerical stability, bilinear stability is more general, applying to any bilinear operators and not just matrix or complex multiplications; is more simplistic, bounding forward error in terms of a single (growth) factor; and is truly tensorial like bilinear complexity, invariant under any orthogonal change of coordinates. As an aside, we study a new algorithm for computing complex multiplication in terms of real, much like Gauss's, but is optimally fast and stable in that it attains both tensor rank and nuclear norm.

Mathematics Subject Classification 65F05 · 14N07 · 46M05

✉ Lek-Heng Lim
lekheng@uchicago.edu

Zhen Dai
zhen9@uchicago.edu

1 Computational and Applied Mathematics Initiative, University of Chicago, Chicago, IL 60637-1514, USA

1 Introduction

More than fifty years ago, in Volume 13 of this journal, Volker Strassen announced an astounding result: A pair of 2×2 matrices may be multiplied with seven multiplications [32]. A consequence is that linear systems can be solved in $O(n^{\log_2 7})$ time complexity, a surprise at that time as existing works such as [23] purportedly showed that $O(n^3)$ was the lowest possible.

Strassen's algorithm is in the spirit of the well-known algorithm, often attributed to Gauss,[1] for multiplying a pair of complex numbers with three real multiplications [17],

$$(a+bi)(c+di) = (ac-bd) + i[(a+b)(c+d) - ac - bd], \tag{1.1}$$

but is notable in that Strassen's applies to a noncommutative product (matrix multiplication) as opposed to a commutative one (complex scalar multiplication). It led to a plethora of followed-up works and ultimately to the realization that there is a unified framework underlying the algorithms of Gauss and Strassen, namely, in evaluating a bilinear operator $\beta\colon \mathbb{U}\times\mathbb{V}\to\mathbb{W}$, viewed as a 3-tensor in $\mathbb{U}^*\otimes\mathbb{V}^*\otimes\mathbb{W}$, any decomposition

$$\beta = \varphi_1\otimes\psi_1\otimes w_1 + \cdots + \varphi_r\otimes\psi_r\otimes w_r \tag{1.2}$$

into linear functionals $\varphi_i : \mathbb{U}\to\mathbb{R}$, $\psi_i : \mathbb{V}\to\mathbb{R}$, and vectors $w_i\in\mathbb{W}$, $i=1,\dots,r$, gives us an algorithm for computing β. Furthermore, the number of terms r in such a decomposition counts precisely the number of multiplications, and thus the minimal value of r, i.e., the *tensor rank* of β, gives the optimal complexity for evaluating β in an appropriate sense [33] (see Sect. 2). Both Gauss's and Strassen's algorithms are the fastest possible according to this measure, that is, they attain the tensor ranks of complex multiplication (three) and 2×2 matrix product (seven) respectively [37].

Well-known to readers of this journal, speed is not all that matters in an algorithm, numerical stability is arguably more important in finite-precision computations as rounding errors may result in an unstable algorithm producing no correct digits. While the stability of algorithms for evaluating bilinear operators has been studied for specific algorithms or operators in isolation, e.g., for Gauss's algorithm in [17], Strassen's algorithm in [7], and other fast matrix multiplication algorithms in [2, 4], there has been no unfied treatment that applies to all bilinear operators β as in the case of speed. There is no analysis that quantifies stability in terms of some tensorial property of β analogous to how speed is quantified in terms of its tensor rank. The goal of the present article is to fill this gap. We will show that just as the number of terms r in the decomposition (1.2) controls the speed of the algorithm, the *growth factor*, defined as

$$\|\varphi_1\|_*\|\psi_1\|_*\|w_1\| + \cdots + \|\varphi_r\|_*\|\psi_r\|_*\|w_r\|, \tag{1.3}$$

controls the stability of the algorithm; and just as the tensor rank of β measures the optimal speed, the *tensor nuclear norm* of β, defined as

[1] See [28, p. 37] [29, p. 8] for example.

$$\|\beta\|_\nu := \inf\left\{\sum_{i=1}^r \|\varphi_i\|_*\|\psi_i\|_*\|w_i\| : \beta = \sum_{i=1}^r \varphi_i \otimes \psi_i \otimes w_i\right\}, \tag{1.4}$$

measures the optimal stability, the precise meaning of which we will state in due course.

Although we have alluded to the relation between tensor nuclear norm and numerical stability in earlier works [15, 27, 38], we have never stated a precise relation nor carried out numerical experiments to demonstrate the relation. This article provides both. Theorem 3.3 gives a general relation between the growth factor of a bilinear algorithm and its forward error, from which a relation between tensor nuclear norm and forward error may be deduced as in Corollary 3.4. We then perform a range of numerical experiments involving Gauss's and Strassen's algorithms to substantiate our theoretical findings:

Matrix multiplication: We compare Strassen's algorithm with a well-known variant due to Winograd [19, 24]. While both attain the optimal seven multiplications, Winograd's variant is often favored because it requires only fifteen additions, compared to Strassen's eighteen. Nevertheless we will show that Strassen's algorithm has a growth factor of $12+2\sqrt{2} \approx 14.83$ whereas Winograd's variant has a growth factor of $7+4\sqrt{2}+3\sqrt{3} \approx 17.85$. For comparison, the conventional algorithm for 2×2 matrix product has eight multiplications and a growth factor of 8. Our numerical experiments confirm that in terms of accuracy Winograd's is indeed worse than Strassen's, which is in turn worse than the conventional algorithm, as Theorem 3.3 indicates.

Complex multiplication: We compare the regular algorithm for complex multiplication, which requires four real multiplications and has a growth factor of 4; Gauss's algorithm, which requires three real multiplications but has a larger growth factor of $2(1+\sqrt{2}) \approx 4.83$; and a new algorithm:

$$\begin{aligned}(a+bi)(c+di) \\ &= \frac{1}{2}\left[\left(a+\frac{1}{\sqrt{3}}b\right)\left(c+\frac{1}{\sqrt{3}}d\right)+\left(a-\frac{1}{\sqrt{3}}b\right)\left(c-\frac{1}{\sqrt{3}}d\right)-\frac{8}{3}bd\right] \\ &\quad + \frac{i\sqrt{3}}{2}\left[\left(a+\frac{1}{\sqrt{3}}b\right)\left(c+\frac{1}{\sqrt{3}}d\right)-\left(a-\frac{1}{\sqrt{3}}b\right)\left(c-\frac{1}{\sqrt{3}}d\right)\right].\end{aligned} \tag{1.5}$$

This new algorithm has the best features of both the regular and Gauss's algorithms, requiring three real multiplications and yet has the smaller (in fact, smallest, as we will see) growth factor of 4. Again the results are consistent with the prediction of Theorem 3.3.

For the uninitiated, we would like to stress that the aforementioned algorithms only begin to make a difference when they are applied recursively, or applied to matrices, or both. For instance, Gauss's algorithm (1.1) is really quite useless for multiplying a pair of complex numbers, whether 'by hand' or on a computer. It only becomes useful when applied recursively in the form of Karatsuba's algorithm [21] for integer multiplication, with i replaced by the number base; or when applied to complex matrices [13]:

$$(A+iB)(C+iD) = (AC-BD) + i[(A+B)(C+D) - AC - BD], \tag{1.6}$$

with $A + iB, C + iD \in \mathbb{C}^{n\times n}$, $A, B, C, D \in \mathbb{R}^{n\times n}$. As multiplication of matrices is much more expensive than addition of matrices, so (1.6) really does represent an enormous savings in speed over the regular algorithm:

$$(A + iB)(C + iD) = (AC - BD) + i(BC + AD). \tag{1.7}$$

Likewise, our new algorithm (1.5) only begins to make a difference when applied to matrices. For the same reason, the algorithms of Strassen and Winograd are only worth the trouble when applied recursively to a product of $n \times n$ matrices partitioned recursively into 2×2 blocks.

To address another related point early on, a surprisingly common complaint among early feedbacks is that there are a lot of $\sqrt{3}$'s in our algorithm (1.5). Certainly, if one computes these products 'by hand,' it would be easier to use the regular or Gauss's algorithm since they do not involve irrational coefficients. But when performed by a computer this is completely immaterial. In case it is not clear, it does not matter whether we multiply by 3 or by $\sqrt{3}$; to a computer (or any IEEE 754-compliant equipment) both are binary strings of 0's and 1's and arithmetic takes one flop regardless. Maybe there would be some minor savings when a constant happens to be a power of 2—because of binary arithmetic—but aside from that, it makes no difference what coefficients appear in our algorithm.

For the matrix multiplication experiments, our goal is to illustrate Theorem 3.3 by comparing the known algorithms of Strassen and Winograd. Incidentally, a numerical comparison of the accuracy of Strassen's algorithm and Winograd's variant was stated as a research problem in [17, Exercise 23.10]. Our work in Sect. 4 supplies both numerical evidence and a rigorous explanation of why Strassen's is more accurate than Winograd's.

For the complex multiplication experiments, aside from providing another illustration of Theorem 3.3, we also have the additional goal of testing, for the first time, the new algorithm (1.5) applied to multiply complex matrices, which we will see is

- nearly as fast as Gauss's algorithm (1.6), and
- nearly as stable as the regular algorithm (1.7).

To substantiate these claims, we perform more extensive experiments to compare (1.5), (1.6), and (1.7), including three practical applications: evaluation of matrix polynomials via Horner's method [20], unitary transform, and complex-valued neural networks [1, 3, 9, 31, 36, 39]. All our codes are available from https://github.com/zhen06/Complex-Matrix-Multiplication.

Conventions

To reduce notational clutter, we denote norms on different vector spaces $\mathbb{U}, \mathbb{V}, \mathbb{W}$ by the same $\|\cdot\|$. There is no cause for confusion since we always use it in a form like $\|v\|$ for some $v \in \mathbb{V}$, where it is clear from context that $\|\cdot\|$ refers to a norm on $\mathbb{V}$. Likewise the corresponding dual norms on $\mathbb{U}^*, \mathbb{V}^*, \mathbb{W}^*$ will be denoted by the same $\|\cdot\|_*$. Recall that for $\varphi \in \mathbb{V}^*$, i.e., $\varphi : \mathbb{V} \to \mathbb{R}$ is a linear functional, this is defined by

$$\|\varphi\|_* := \sup\{|\varphi(v)| : \|v\| \le 1\}.$$

In this article, "stability" and "accuracy" have the same meaning, i.e., small forward error, but the former is used to describe an algorithm whereas the latter is used to describe its output.

2 Bilinear complexity

We provide a brief review of bilinear complexity, usually studied in Algebraic Computational Complexity [6, 8, 26, 35], for numerical analysts. Our goals here are to (i) highlight certain departures from typical practice in numerical linear algebra; and (ii) show a parallel with our notion of bilinear stability in the next section.

Let $\mathbb{U}, \mathbb{V}, \mathbb{W}$ be finite-dimensional vector spaces, assume to be over $\mathbb{R}$ for simplicity. Let $\beta : \mathbb{U} \times \mathbb{V} \to \mathbb{W}$ be a bilinear operator. Depending on one's definition of a tensor, we have $\beta \in \mathbb{U}^* \otimes \mathbb{V}^* \otimes \mathbb{W}$ either through definition [27, Definition 3.3] or by the universal mapping property [27, Equation 4.88]. A *bilinear algorithm* for evaluating β is a decomposition of the form (1.2). In other words, for any $u \in \mathbb{U}$ and $v \in \mathbb{V}$, we evaluate $\beta(u, v)$ by performing the algorithm given by the decomposition on the right:

$$\beta(u, v) = \sum_{i=1}^{r} \varphi_i(u)\psi_i(v)w_i. \tag{2.1}$$

In practice, the vector spaces involved are usually Euclidean spaces of vectors $\mathbb{R}^n$ or matrices $\mathbb{R}^{m\times n}$. Riesz representation theorem guarantees that any linear functional $\varphi : \mathbb{R}^n \to \mathbb{R}$ must take the form $\varphi(x) = a^{\mathsf{T}}x$ for some $a \in \mathbb{R}^n$ and likewise any functional $\varphi : \mathbb{R}^{m\times n} \to \mathbb{R}$ must take the form $\varphi(X) = \operatorname{tr}(A^{\mathsf{T}}X)$ for some $A \in \mathbb{R}^{m\times n}$.

Each rank-one term $\varphi_i(u)\psi_i(v)w_i$ in (2.1) accounts for one multiplication but herein lies a pitfall — the 'multiplication' refers to the product of $\varphi_i(u)$ and $\psi_i(v)$; note that this a variable product, i.e., the value depends on variables u and v, as opposed to a scalar product. Take a randomly made-up example[2] with $\mathbb{U} = \mathbb{R}^{2\times 2}$, $\mathbb{V} = \mathbb{R}^2$, $\mathbb{W} = \mathbb{R}^3$, and

$$\varphi_i\left(\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]\right) = \operatorname{tr}\left(\left[\begin{smallmatrix} -1 & 0 \\ 1 & 2 \end{smallmatrix}\right]^{\mathsf{T}}\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]\right) = -a + c + 2d,$$
$$\psi_i\left(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]\right) = \left[\begin{smallmatrix} 3 \\ -1/2 \end{smallmatrix}\right]^{\mathsf{T}}\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = 3x - y/2, \quad w_i = \begin{bmatrix} -3 \\ 4 \\ \sqrt{5} \end{bmatrix},$$

then there is exactly one multiplication in

$$\varphi_i(u)\psi_i(v)w_i = \begin{bmatrix} -3(-a+c+2d)(3x-y/2) \\ 4(-a+c+2d)(3x-y/2) \\ \sqrt{5}(-a+c+2d)(3x-y/2) \end{bmatrix}.$$

The scalar products like $2d$ or $-y/2$ or $\sqrt{5}t$ are discounted in Strassen's model of bilinear complexity [33, 34] and for good reasons — these constants coefficients are fixed in the algorithm and can be hardcoded or hardwired, unlike the product between

[2] Genuine examples to follow in Sects. 4 and 5.

$-a+c+2d$ and $3x-y/2$, which depends on the variable inputs $u=\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ and $v=\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$. In particular, Strassen's measure of speed, called *bilinear complexity*, is independent of the values of these constant coefficients, but we will show in the next section that these will affect numerical stability of the algorithm.

To emphasize its distinction from scalar products, Strassen calls a variable product in the above sense a *nonscalar product* [34]. In other words, bilinear complexity measures speed purely in terms of the number of nonscalar products. The bilinear complexity of the algorithm in (2.1) is given by the number terms in the decomposition r and the optimal speed of evaluating β is therefore given by the tensor rank [33]

$$\operatorname{rank}(\beta) := \min\left\{ r : \beta = \sum_{i=1}^{r} \varphi_i \otimes \psi_i \otimes w_i \right\}. \tag{2.2}$$

A *tensor rank decomposition* of β, i.e., one that attains its tensor rank, is then a fastest algorithm in the context of bilinear complexity.

In realistic scenarios, storage and computations both have finite-precision. Given u and v, we do not need to know $\beta(u, v)$ exactly; in fact computing anything beyond 16 decimal digits of accuracy is wasted effort since we do not store more than 16 digits in IEEE double precision. So the tensor rank of β is less relevant than the *border rank* [5] of β, which is the smallest r so that

$$\|\beta - \varphi_1^{\varepsilon} \otimes \psi_1^{\varepsilon} \otimes w_1^{\varepsilon} - \varphi_2^{\varepsilon} \otimes \psi_2^{\varepsilon} \otimes w_2^{\varepsilon} - \cdots - \varphi_r^{\varepsilon} \otimes \psi_r^{\varepsilon} \otimes w_r^{\varepsilon}\| < \varepsilon$$

for all $\varepsilon > 0$, or, formally,

$$\overline{\operatorname{rank}}(\beta) := \min\left\{ r : \beta = \lim_{\varepsilon \to 0^+} \sum_{i=1}^{r} \varphi_i^{\varepsilon} \otimes \psi_i^{\varepsilon} \otimes w_i^{\varepsilon} \right\}. \tag{2.3}$$

For the two problems studied in our article, namely, matrix multiplication,

$$\beta_{m,n,p} : \mathbb{R}^{m\times n} \times \mathbb{R}^{n\times p} \to \mathbb{R}^{m\times p}, \quad (A, B) \mapsto AB,$$

and complex multiplication,

$$\beta_{\mathbb{C}} : \mathbb{C} \times \mathbb{C} \to \mathbb{C}, \quad (w, z) \mapsto wz,$$

(noting that $\mathbb{C}$ is a two-dimensional real vector space), we have [25, 37]

$$\operatorname{rank}(\beta_{2,2,2}) = \overline{\operatorname{rank}}(\beta_{2,2,2}) = 7, \qquad \operatorname{rank}(\beta_{\mathbb{C}}) = \overline{\operatorname{rank}}(\beta_{\mathbb{C}}) = 3.$$

It is in general difficult to find such exact values. For instance, the values of $\operatorname{rank}(\beta_{3,3,3})$ and $\overline{\operatorname{rank}}(\beta_{3,3,3})$ are still unknown. Most of the efforts in studying matrix multiplication go towards determining the asymptotic value $\omega := \inf\{p \in \mathbb{R} \colon \operatorname{rank}(\beta_{n,n,n}) = O(n^p)\}$, called the *exponent of matrix multiplication*. An advantage is that asymptotically, the full arithmetic complexity, i.e., counting all operations and not just nonscalar multiplications, is also $O(n^\omega)$. More importantly, the role of ω stretches far beyond matrix multiplication, governing the full arithmetic complexity of computing inverse, determinant, null basis, linear systems, LU/QR/eigenvalue/Hessenberg decompositions, characteristic polynomials, sparsification, and even linear programming — note in particular that none of these are bilinear operations [34] (see also [8, Chapter 16] and [27, Examples 3.10 and 4.40].

3 Bilinear stability

We would like to state at the outset that numerical stability is a moderately complicated issue that depends on many factors and cannot be completely represented by any single number. Designing numerically stable algorithms is as much an art as it is a science. However the six *Higham guidelines* for numerical stability [19, Section 1.18] capture the most salient aspects. Among them, the second guideline to "minimize the size of intermediate quantities relative to the final solution" is one of the most unequivocal, lends itself to precise quantification, and is what we will focus on in this section. Consideration of Higham's second guideline for bilinear algorithms leads us naturally to the notion of *bilinear stability*, which relates to accuracy the way bilinear complexity relates to speed. More precisely, the growth factor (1.3) and tensor nuclear norm (1.4) are to accuracy in bilinear stability what the number of rank-1 terms in (2.1) and the tensor rank (2.2) are to speed in bilinear complexity. Here accuracy refers to the size of relative forward error.

Bilinear stability differs from existing studies of numerical stability of bilinear algorithms such as those in [2, 4, 7, 17] in three ways: (i) it is more general, applying to any bilinear operators as opposed to specific ones like matrix multiplication; (ii) it is more simplistic, relating forward error to just growth factor as opposed to two or three different factors in the approaches of [2, 4]; (iii) it is truly tensorial, as growth factor and tensor nuclear norm are invariant under any orthogonal change-of-coordinates, just as tensor rank is invariant under any invertible change-of-coordinates. The factors (i) and (ii), i.e., generality and simplicity, may often be sacrificed for better bounds: Given any specific bilinear operator, we may often obtain smaller forward error bounds by performing a more precise analysis tailored to that given operator. We will do see this in Sect. 5.2.

One difference between bilinear complexity and bilinear stability is that the latter requires a norm. While there are many excellent treatises on tensor norms [10, 12, 30], they are excessive for our purpose. All the reader needs to know is that for a vector space $\mathbb{V}_i$ with norm $\|\cdot\|_i$, $i = 1, \dots, d$, a tensor norm $\|\cdot\|$ on $\mathbb{V}_1 \otimes \mathbb{V}_2 \otimes \cdots \otimes \mathbb{V}_d$ satisfies the multiplicativity property for rank-1 tensors:

$$\|v_1 \otimes v_2 \otimes \cdots \otimes v_d\| = \|v_1\|_1 \|v_2\|_2 \cdots \|v_d\|_d,$$

where $v_i \in \mathbb{V}_i$. In particular, the spectral, Frobenius (also called Hilbert–Schmidt), nuclear norms [27, p. 561 and Example 4.17] are all equal on rank-1 tensors in $\mathbb{U}^* \otimes$

$\mathbb{V}^* \otimes \mathbb{W}$, i.e.,

$$\|\varphi \otimes \psi \otimes w\|_\sigma = \|\varphi \otimes \psi \otimes w\|_{\mathsf{F}} = \|\varphi \otimes \psi \otimes w\|_\nu = \|\varphi\|_*\|\psi\|_*\|w\|$$

for all $\varphi \in \mathbb{U}^*$, $\psi_i \in \mathbb{V}^*$, $w \in \mathbb{W}$. Consequently, when we speak of the norm of a rank-1 tensor $\varphi \otimes \psi \otimes w$, it does not matter which of these three norms we choose, and we will simply write

$$\|\varphi \otimes \psi \otimes w\| := \|\varphi\|_*\|\psi\|_*\|w\|.$$

We first present a straightforward heurstic that motivates our definition of the growth factor, deferring the more formal forward error analysis to Theorem 3.3. If we apply the rank-one bilinear operator $\varphi_i \otimes \psi_i \otimes w_i$ to u and v,

$$\begin{aligned}\|(\varphi_i \otimes \psi_i \otimes w_i)(u, v)\| &= \|\varphi_i(u)\psi_i(v)w_i\| = |\varphi_i(u)||\psi_i(v)|\|w_i\| \\ &\le \|\varphi_i\|_*\|u\|\|\psi_i\|_*\|v\|\|w_i\| = \|\varphi_i \otimes \psi_i \otimes w_i\|\|u\|\|v\|.\end{aligned}$$

So $\varphi_i \otimes \psi_i \otimes w_i$ magnifies the errors in u and v by an amount bounded by its tensor norm $\|\varphi_i \otimes \psi_i \otimes w_i\|$. Therefore, in a bilinear algorithm given by the right side of (2.1) for evaluating β, triangle inequality gives

$$\|\beta(u, v)\| = \left\|\sum_{i=1}^r (\varphi_i \otimes \psi_i \otimes w_i)(u, v)\right\| \le \left[\sum_{i=1}^r \|\varphi_i \otimes \psi_i \otimes w_i\|\right]\|u\|\|v\|.$$

The ith step of the algorithm magnifies the error in the inputs (u, v) by an amount bounded by $\|\varphi_i \otimes \psi_i \otimes w_i\|$ and over the course of r steps in the algorithm, the accumulated error is bounded by a factor of

$$\sum_{i=1}^r \|\varphi_i \otimes \psi_i \otimes w_i\| = \sum_{i=1}^r \|\varphi_i\|_*\|\psi_i\|_*\|w_i\|, \tag{3.1}$$

which we will define as the *growth factor* of the algorithm or decomposition (2.1). Its minimum value over all possible bilinear algorithms for evaluating β or, equivalently, over all decomposition of β as a 3-tensor is therefore given by the nuclear norm (1.4). This idea was first floated in [38, Section 3.2]. Note that the growth factor depends on the algorithm/decomposition for β but the nulcear norm depends only on β.

We now state a formal definition to make precise the terms used in the preceding discussions.

Definition 3.1 Let $\mathbb{U}, \mathbb{V}, \mathbb{W}$ be three finite-dimensional real vector spaces. A *decomposition* of a bilinear operator $\beta : \mathbb{U} \times \mathbb{V} \to \mathbb{W}$ is a list $\mathcal{D} = (\varphi_i, \psi_i, w_i)_{i=1}^r$ with

$$\beta = \sum_{i=1}^r \varphi_i \otimes \psi_i \otimes w_i, \tag{3.2}$$

where $\varphi_i : \mathbb{U} \to \mathbb{R}$ and $\psi_i : \mathbb{V} \to \mathbb{R}$ are linear functionals and $w_i \in \mathbb{W}$, $i = 1, \dots, r$. An *algorithm* $\widehat{\beta}_{\mathcal{D}}$ given by the decomposition $\mathcal{D}$ takes $(u, v) \in \mathbb{U} \times \mathbb{V}$ as inputs and computes the output $\beta(u, v)$ in three steps:

(i) computes $\varphi_i(u)$ and $\psi_i(v)$, $i = 1, \dots, r$;
(ii) computes $\varphi_i(u)\psi_i(v)w_i$, $i = 1, \dots, r$;
(iii) computes $\sum_{i=1}^{r} \varphi_i(u)\psi_i(v)w_i$.

The *growth factor* of the algorithm $\widehat{\beta}_{\mathcal{D}}$ is defined as

$$\gamma(\widehat{\beta}_{\mathcal{D}}) := \sum_{i=1}^{r} \|\varphi_i \otimes \psi_i \otimes w_i\| = \sum_{i=1}^{r} \|\varphi_i\|_* \|\psi_i\|_* \|w_i\|.$$

As noted in Sect. 2, only the variable multiplication in step (ii) counts in bilinear complexity; the other two steps comprising scalar multiplications and additions are discounted. In bilinear stability all three steps contribute to the growth factor.

Proposition 3.2 *The minimal growth factor is given by nucler norm of the β, i.e.,*

$$\min_{\mathcal{D}} \gamma(\widehat{\beta}_{\mathcal{D}}) = \|\beta\|_{\nu},$$

with $\mathcal{D}$ running over all decomposition. Furthermore, there is always an algorithm that attains the minimal growth factor.

The above equality is just stating (1.4) in terms of the growth factor. That there is always an algorithm attaining the minimal growth factor, justifying our writing min instead of inf, follows from the existence of a *nuclear decomposition* [15, Proposition 3.1], i.e., a decomposition that attains the nuclear norm. Just as a rank decomposition of β represents a fastest algorithm in bilinear complexity, a nuclear decomposition of β represents a stablest algorithm in bilinear stability.

We next establish a rigorous relationship between growth factor and numerical stability by proving a forward error bound in terms of the growth factor of a bilinear algorithm. We assume a system of floating point arithmetic obeying the standard model as in [19]: For $x, y \in \mathbb{R}$

$$\mathsf{fl}(x \ \mathsf{op}\ y) = (x \ \mathsf{op}\ y)(1 + \delta), \qquad |\delta| \le \mathbf{u}, \quad \mathsf{op} = +, -, *, / \tag{3.3}$$

with $\mathbf{u}$ the unit roundoff, except when $\mathsf{fl}(x \ \mathsf{op}\ y) = 0$, in which case δ becomes -1. We assume that $\mathbb{U}, \mathbb{V}, \mathbb{W}$ are vector spaces of dimensions m, n, p and that appropriate computational bases have been chosen on them so that we may identify $\mathbb{U} \cong \mathbb{R}^m$, $\mathbb{V} \cong \mathbb{R}^n$, $\mathbb{W} \cong \mathbb{R}^p$. The computational bases do not need to be the standard bases and may instead be Fourier, Krylov, Haar, wavelet bases, etc. This is another reason why we cast our discussions in terms of abstract vector spaces and do not choose bases until absolutely necessary. However, once a choice of bases has been made, the result below depends only on the dimensions of $\mathbb{U}, \mathbb{V}, \mathbb{W}$; if say, $\mathbb{U} = \mathbb{R}^{m \times n}$, then only the fact that it has dimension mn matters, i.e., $\mathbb{U} \cong \mathbb{R}^{mn}$.

Theorem 3.3 (Growth factor and forward error) *Let $\beta : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^p$ be a bilinear operator, $\mathcal{D} = (\varphi_i, \psi_i, w_i)_{i=1}^r$ a decomposition, and $\widehat{\beta}_{\mathcal{D}}$ the corresponding algorithm. If $\widehat{\beta}_{\mathcal{D}}(u, v)$ is the output of $\widehat{\beta}_{\mathcal{D}}$ computed using floating point operations, with $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ as inputs, then*

$$\|\beta(u, v) - \widehat{\beta}_{\mathcal{D}}(u, v)\|_\infty \le (m + n + r + 1)\gamma(\widehat{\beta}_{\mathcal{D}})\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2).$$

Proof We first show that the result reduces to the case $p = 1$. It suffices to show that

$$|\beta(u, v)_k - \widehat{\beta}_{\mathcal{D}}(u, v)_k| \le (m + n + r + 1)\gamma(\widehat{\beta}_{\mathcal{D}})\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2) \tag{3.4}$$

for all $k = 1, \ldots, p$, where the subscript k refers to the kth coordinate of a vector in $\mathbb{R}^p$. Since

$$\gamma(\widehat{\beta}_{\mathcal{D}}) = \sum_{i=1}^r \|\varphi_i\|_* \|\psi_i\|_* \|w_i\| \ge \sum_{i=1}^r \|\varphi_i\|_* \|\psi_i\|_* |w_{ik}|,$$

with w_{ik} the kth coordinate of $w_i \in \mathbb{R}^p$, to show (3.4), it suffices to show

$$|\beta(u, v)_k - \widehat{\beta}_{\mathcal{D}}(u, v)_k| \le (m + n + r + 1)\left[\sum_{i=1}^r \|\varphi_i\|_* \|\psi_i\|_* |w_{ik}|\right]\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2),$$

which is equivalent to the case $p = 1$. In the following, we will assume that $p = 1$.

Since φ_i and ψ_i are linear functionals on $\mathbb{R}^m$ and $\mathbb{R}^n$, there exist $u_i \in \mathbb{R}^m$ and $v_i \in \mathbb{R}^n$ such that

$$\varphi_i(u) = u_i^\mathsf{T} u \quad \text{and} \quad \psi_i(v) = v_i^\mathsf{T} v,$$

for all $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$. By [19, equation 3.7],

$$|x^\mathsf{T} y - \mathsf{fl}(x^\mathsf{T} y)| \le n|x|^\mathsf{T}|y|\mathbf{u} + O(\mathbf{u}^2),$$

for any $x, y \in \mathbb{R}^n$ where $|\cdot|$ applies coordinatewise. So for each $i = 1, \ldots, r$,

$$\begin{aligned} |\varphi_i(u) - \mathsf{fl}(\varphi_i(u))| &= |u_i^\mathsf{T} u - \mathsf{fl}(u_i^\mathsf{T} u)| \le m|u_i|^\mathsf{T}|u|\mathbf{u} + O(\mathbf{u}^2) \\ &\le m\|u_i\|\|u\|\mathbf{u} + O(\mathbf{u}^2) = m\|\varphi_i\|_*\|u\|\mathbf{u} + O(\mathbf{u}^2). \end{aligned} \tag{3.5}$$

Likewise, for each $i = 1, \ldots, r$,

$$|\psi_i(v) - \mathsf{fl}(\psi_i(v))| \le n\|\psi_i\|_*\|v\|\mathbf{u} + O(\mathbf{u}^2). \tag{3.6}$$

Let $\Delta_{1,i} = \mathsf{fl}(\varphi_i(u)) - \varphi_i(u)$ and $\Delta_{2,i} = \mathsf{fl}(\psi_i(v)) - \psi_i(v)$. By (3.5) and (3.6),

$$|\Delta_{1,i}| \le m\|\varphi_i\|_*\|u\|\mathbf{u} + O(\mathbf{u}^2), \quad |\Delta_{2,i}| \le n\|\psi_i\|_*\|v\|\mathbf{u} + O(\mathbf{u}^2). \tag{3.7}$$

Let $c_i = \varphi_i(u)\psi_i(v)$ and $\widehat{c_i}$ be its computed value. By (3.7), there exists δ_i with $|\delta_i| \le \mathbf{u}$ such that

$$\begin{aligned}\widehat{c_i} &= (\varphi_i(u) + \Delta_{1,i})(\psi_i(v) + \Delta_{2,i})(1+\delta_i) \\ &= \varphi_i(u)\psi_i(v) + \Delta_{1,i}\psi_i(v) + \varphi_i(u)\Delta_{2,i} + \delta_i\varphi_i(u)\psi_i(v) + O(\mathbf{u}^2). \end{aligned} \tag{3.8}$$

By (3.7) and (3.8),

$$\begin{aligned}|c_i - \widehat{c_i}| &\le m\|\varphi_i\|_*\|u\||\psi_i(v)|\mathbf{u} + |\varphi_i(u)|n\|\psi_i\|_*\|v\|\mathbf{u} + |\varphi_i(u)\psi_i(v)|\mathbf{u} + O(\mathbf{u}^2) \\ &\le (m+n+1)\|\varphi_i\|_*\|\psi_i\|_*\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2). \end{aligned} \tag{3.9}$$

Let $\Delta_i = \widehat{c_i} - c_i$. By (3.9),

$$|\Delta_i| \le (m+n+1)\|\varphi_i\|_*\|\psi_i\|_*\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2). \tag{3.10}$$

Let $d_i = c_i w_i$ and $\widehat{d_i}$ be the computed value of d_i. By (3.10), there exists δ_i' with $|\delta_i'| \le \mathbf{u}$ such that

$$\widehat{d_i} = (c_i + \Delta_i)w_i(1+\delta_i') = c_i w_i + \Delta_i w_i + \delta_i' c_i w_i + O(\mathbf{u}^2). \tag{3.11}$$

Let $\Delta_i' = \widehat{d_i} - d_i$. By (3.10) and (3.11),

$$\begin{aligned}|\Delta_i'| &\le (m+n+1)\|\varphi_i\|_*\|\psi_i\|_*\|u\|\|v\||w_i|\mathbf{u} + |\varphi_i(u)\psi_i(v)||w_i|\mathbf{u} + O(\mathbf{u}^2) \\ &\le (m+n+2)\|\varphi_i\|_*\|\psi_i\|_*|w_i|\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2). \end{aligned} \tag{3.12}$$

Finally, let $a = \sum_{i=1}^r \varphi_i(u)\psi_i(v)w_i$ and $\widehat{a}$ be the computed value of a. By (3.12), there exists δ with $|\delta| \le \mathbf{u}$ such that

$$\widehat{a} = \widehat{d}_1(1+\delta)^{r-1} + \widehat{d}_2(1+\delta)^{r-1} + \widehat{d}_3(1+\delta)^{r-2} + \cdots + \widehat{d}_r(1+\delta),$$

where we compute the sum $\widehat{d}_1 + \widehat{d}_2 + \cdots + \widehat{d}_r$ from left to right. Hence we obtain

$$\begin{aligned}|a - \widehat{a}| &\le (m+n+2)\|u\|\|v\|\sum_{i=1}^r \|\varphi_i\|_*\|\psi_i\|_*|w_i|\mathbf{u} + (r-1)\left\|\sum_{i=1}^r c_i w_i\right\|\mathbf{u} + O(\mathbf{u}^2) \\ &\le (m+n+r+1)\|u\|\|v\|\sum_{i=1}^r \|\varphi_i\|_*\|\psi_i\|_*|w_i|\mathbf{u} + O(\mathbf{u}^2) \\ &= (m+n+r+1)\gamma(\widehat{\beta_{\mathcal{D}}})\|u\|\|v\|\mathbf{u} + O(\mathbf{u}^2). \end{aligned}$$

□

Theorem 3.3 essentially says that that algorithms with small growth factors have small forward errors. Combined with Proposition 3.2, we see that the optimally stable algorithm in this context is the one corresponding to a nuclear decomposition of β.

Corollary 3.4 (Tensor nuclear norm and forward error) *Let $\beta : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$ be a bilinear operator, $\mathcal{D} = (\varphi_i, \psi_i, w_i)_{i=1}^r$ a nuclear decomposition, and $\widehat{\beta}_{\mathcal{D}}$ the corresponding algorithm. Then*

$$\|\beta(u, v) - \widehat{\beta}_{\mathcal{D}}(u, v)\|_\infty \le (m + n + r + 1)\|\beta\|_\nu \|u\| \|v\| \mathbf{u} + O(\mathbf{u}^2).$$

In principle, there is no reason to expect there to be an algorithm that is both fastest in the sense of Sect. 2 and stablest in the sense of this section, i.e., having a decomposition that attains both tensor rank and nuclear norm. In Sect. 5, we will see that such an algorithm exists for complex multiplication and we will study its properties when applied to complex matrix multiplication.

4 Fast matrix multiplications

As an illustration of bilinear stability in the last section, we will calculate the growth factors of Strassen's algorithm [32] and Winograd's variant [19, 24] for fast matrix multiplication and compare their stability empirically. We will see that the growth factor of Strassen's algorithm is smaller than that of Winograd's variant, and, consistent with the prediction of Theorem 3.3, numerical experiments indeed show that the former gives more accurate results.

4.1 Bilinear stability of Strassen multiplication

Given two block matrices

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

Strassen's algorithm [32] first computes

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), & M_5 &= (A_{11} + A_{12})B_{22}, \\ M_2 &= (A_{21} + A_{22})B_{11}, & M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ M_3 &= A_{11}(B_{12} - B_{22}), & M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \\ M_4 &= A_{22}(B_{21} - B_{11}), \end{aligned}$$

and then computes the product via

$$AB = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}.$$

Note that this may be applied recursively. Let $\widehat{\beta_{\mathsf{S}}} : \mathbb{R}^{2\times 2} \times \mathbb{R}^{2\times 2} \to \mathbb{R}^{2\times 2}$ denote the Strassen's algorithm for 2×2 matrices. It is routine to check that for $A, B \in \mathbb{R}^{2\times 2}$,

$$\widehat{\beta_{\mathsf{S}}}(A, B) = \sum_{i=1}^{7} \varphi_i(A)\psi_i(B)W_i,$$

where $\varphi_i(A) = \operatorname{tr}(U_i^{\mathsf{T}} A)$ and $\psi_i(B) = \operatorname{tr}(V_i^{\mathsf{T}} B)$ with

$$\begin{aligned}
U_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & V_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & W_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \\
U_2 &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, & V_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & W_2 &= \begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}; \\
U_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & V_3 &= \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & W_3 &= \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}; \\
U_4 &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & V_4 &= \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, & W_4 &= \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}; \\
U_5 &= \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, & V_5 &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & W_5 &= \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}; \\
U_6 &= \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, & V_6 &= \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, & W_6 &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}; \\
U_7 &= \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & V_7 &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, & W_7 &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.
\end{aligned}$$

For simplicity we will use the Frobenius norm on $\mathbb{R}^{2\times 2}$ since it is self dual. The growth factor of Strassen's algorithm is then given by

$$\begin{aligned}
\gamma(\widehat{\beta_{\mathsf{S}}}) = \sum_{i=1}^{7} \|\varphi_i\|_* \|\psi_i\|_* \|W_i\| &= \sum_{i=1}^{7} \|U_i\|_{\mathsf{F}} \|V_i\|_{\mathsf{F}} \|W_i\|_{\mathsf{F}} \\
&= 12 + 2\sqrt{2} \approx 14.83. \qquad (4.1)
\end{aligned}$$

4.2 Bilinear stability of Winograd multiplication

Winograd's algorithm [19, 24] computes a different set of intermediate quantities

$$\begin{aligned}
M_1' &= (A_{21} + A_{22} - A_{11})(B_{11} + B_{22} - B_{12}), & M_5' &= (A_{21} + A_{22})(B_{12} - B_{11}), \\
M_2' &= A_{11}B_{11}, & M_6' &= (A_{11} + A_{12} - A_{21} - A_{22})B_{22}, \\
M_3' &= A_{12}B_{21}, & M_7' &= A_{22}(B_{11} + B_{22} - B_{12} - B_{21}), \\
M_4' &= (A_{11} - A_{21})(B_{22} - B_{12}), & &
\end{aligned}$$

and then compute the product via

$$AB = \begin{bmatrix} M_2' + M_3' & M_1' + M_2' + M_5' + M_6' \\ M_1' + M_2' + M_4' - M_7' & M_1' + M_2' + M_4' + M_5' \end{bmatrix}.$$

Again this can be applied recursively. Let $\widehat{\beta}_{\mathsf{W}} : \mathbb{R}^{2\times 2} \times \mathbb{R}^{2\times 2} \to \mathbb{R}^{2\times 2}$ denote the Winograd's algorithm for 2×2 matrices. It is again routine to check that for $A, B \in \mathbb{R}^{2\times 2}$,

$$\widehat{\beta}_{\mathsf{W}}(A, B) = \sum_{i=1}^{7} \varphi_i'(A)\psi_i'(B)W_i',$$

where $\varphi_i'(A) = \operatorname{tr}(U_i'^{\mathsf{T}} A)$ and $\psi_i'(B) = \operatorname{tr}(V_i'^{\mathsf{T}} B)$ with

$$\begin{aligned}
U_1' &= \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}, & V_1' &= \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}, & W_1' &= \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}; \\
U_2' &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & V_2' &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & W_2' &= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}; \\
U_3' &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, & V_3' &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, & W_3' &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}; \\
U_4' &= \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}, & V_4' &= \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}, & W_4' &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}; \\
U_5' &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, & V_5' &= \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, & W_5' &= \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}; \\
U_6' &= \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, & V_6' &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & W_6' &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}; \\
U_7' &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & V_7' &= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, & W_7' &= \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}.
\end{aligned}$$

With respect to the Frobenius norm, the growth factor of Winograd's algorithm is

$$\begin{aligned}
\gamma(\widehat{\beta}_{\mathsf{W}}) = \sum_{i=1}^{7} \|\varphi_i'\|_* \|\psi_i'\|_* \|W_i'\|_{\mathsf{F}} &= \sum_{i=1}^{7} \|U_i'\|_{\mathsf{F}} \|V_i'\|_{\mathsf{F}} \|W_i'\|_{\mathsf{F}} \\
&= 7 + 4\sqrt{2} + 3\sqrt{3} \approx 17.85. \qquad (4.2)
\end{aligned}$$

4.3 Bilinear stability of conventional matrix multiplication

For completeness we state the growth factor of the conventional algorithm for matrix multiplication $\widehat{\beta}_{\mathsf{C}} : \mathbb{R}^{2\times 2} \times \mathbb{R}^{2\times 2} \to \mathbb{R}^{2\times 2}$,

$$\widehat{\beta}_{\mathsf{C}}(A, B) = \sum_{i,j,k=1}^{2} \operatorname{tr}(E_{ij}^{\mathsf{T}} A) \operatorname{tr}(E_{jk}^{\mathsf{T}} B) E_{ik},$$

where $E_{ij} \in \mathbb{R}^{2\times 2}$ denotes the standard basis matrix. Its growth factor is easily seen to be

$$\gamma(\widehat{\beta}_{\mathsf{C}}) = \sum_{i,j,k=1}^{2} \|E_{ij}\|_{\mathsf{F}} \|E_{jk}\|_{\mathsf{F}} \|E_{ik}\|_{\mathsf{F}} = 8.$$

From (4.1) and (4.2), we see that

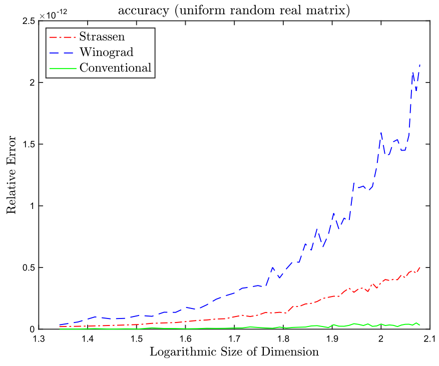
$$\gamma(\widehat{\beta}_{\mathsf{W}}) > \gamma(\widehat{\beta}_{\mathsf{S}}) > \gamma(\widehat{\beta}_{\mathsf{C}}). \tag{4.3}$$

The first inequality will be verified in the numerical experiments below; the second is consistent with the well-known fact [19] that Strassen's algorithm is less stable than conventional multiplication. In this case, the conventional algorithm attains the nuclear norm of two by two matrix multiplication, which has value 8 [11].

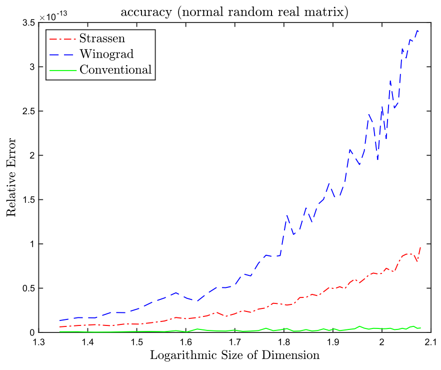
4.4 Numerical experiments for fast matrix multiplications

By Theorem 3.3 and the sizes of the growth factors in (4.3), we expect Strassen's algorithm to give more accurate results than Winograd's variant since it has a smaller growth factor. We test this statement with random matrices generated in three different ways: with (a) real entries drawn from the uniform distribution on $[-1, 1]$, (b) real entries drawn from the standard normal distribution, (c) complex entries whose real and imaginary parts are drawn from the uniform distribution on $[-1, 1]$. In the last case, note that our earlier discussions over $\mathbb{R}$ apply verbatim over $\mathbb{C}$ with the same growth factors.

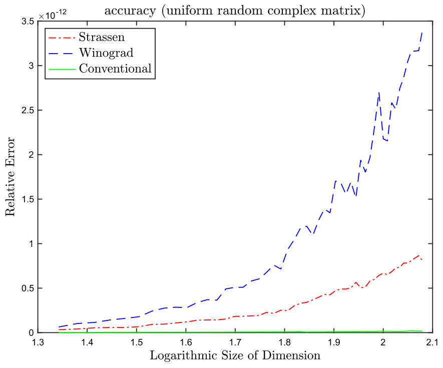
In all cases, we compute $\widehat{\beta}_{\mathsf{S}}(A, B)$ and $\widehat{\beta}_{\mathsf{W}}(A, B)$ using Strassen's algorithm and Winograd's variant respectively and compare the results against the exact value $\beta(A, B) = AB$ computed using the MATLAB symbolic toolbox. From Fig. 1, we see that Strassen's algorithm is indeed more stable than Winograd's variant, substantiating Theorem 3.3. Even though the 14.83 growth factor of Strassen's algorithm appears to differ only moderately from the 17.85 growth factor of Winograd's variant, the effect is magnified multifold as a result of recursion — these algorithms are applied recursively to an $n \times n$ matrix as a block 2×2 matrix $\lfloor \log_2 n \rfloor$ times. The conventional algorithm, which has a growth factor of 8, is included in these plots for comparison.



(A) Real random matrices $\mathcal{U}[-1,1]$.



(B) Real random matrices $\mathcal{N}(0,1)$.



(C) Complex random matrices $\mathcal{U}[-1,1]+\mathcal{U}[-1,1]i$.

Fig. 1 Accuracy of Strassen's algorithm and Winograd's variant

5 Complex multiplication

As described towards the end of Sect. 2, complex multiplication is an $\mathbb{R}$-bilinear operator $\beta_{\mathbb{C}} \in \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$ when we identify $\mathbb{C} \cong \mathbb{R}^2$, with the standard basis vectors in $\mathbb{R}^2$

$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

corresponding to $1, i \in \mathbb{C}$. We write $e_1^*, e_2^* : \mathbb{R}^2 \to \mathbb{R}$ for the dual basis, i.e., linear functionals with

$$e_1^*\left(\begin{bmatrix} a \\ b \end{bmatrix}\right) = a, \qquad e_2^*\left(\begin{bmatrix} a \\ b \end{bmatrix}\right) = b.$$

We will denote the regular algorithm $(a+bi)(c+di) = (ac-bd)+i(bc+ad)$, Gauss's algorithm (1.1), and our new algorithm (1.5) by $\widehat{\beta}_{\mathsf{R}}$, $\widehat{\beta}_{\mathsf{G}}$, $\widehat{\beta}_{\mathsf{N}}$ respectively. For easy reference,

$$\widehat{\beta}_{\mathsf{R}}\left(\begin{bmatrix}a\\b\end{bmatrix},\begin{bmatrix}c\\d\end{bmatrix}\right)=\begin{bmatrix}ac-bd\\bc+ad\end{bmatrix},\qquad \widehat{\beta}_{\mathsf{G}}\left(\begin{bmatrix}a\\b\end{bmatrix},\begin{bmatrix}c\\d\end{bmatrix}\right)=\begin{bmatrix}ac-bd\\(a+b)(c+d)-ac-bd\end{bmatrix},$$
$$\widehat{\beta}_{\mathsf{N}}\left(\begin{bmatrix}a\\b\end{bmatrix},\begin{bmatrix}c\\d\end{bmatrix}\right)=\begin{bmatrix}\frac{1}{2}\left[\left(a+\frac{1}{\sqrt{3}}b\right)\left(c+\frac{1}{\sqrt{3}}d\right)+\left(a-\frac{1}{\sqrt{3}}b\right)\left(c-\frac{1}{\sqrt{3}}d\right)-\frac{8}{3}bd\right]\\ \frac{i\sqrt{3}}{2}\left[\left(a+\frac{1}{\sqrt{3}}b\right)\left(c+\frac{1}{\sqrt{3}}d\right)-\left(a-\frac{1}{\sqrt{3}}b\right)\left(c-\frac{1}{\sqrt{3}}d\right)\right]\end{bmatrix}.$$

They correspond to the decompositions

$$\widehat{\beta}_{\mathsf{R}}=(e_1^*\otimes e_1^*-e_2^*\otimes e_2^*)\otimes e_1+(e_1^*\otimes e_2^*+e_2^*\otimes e_1^*)\otimes e_2, \tag{5.1}$$
$$\widehat{\beta}_{\mathsf{G}}=(e_1^*+e_2^*)\otimes(e_1^*+e_2^*)\otimes e_2+e_1^*\otimes e_1^*\otimes(e_1-e_2)-e_2^*\otimes e_2^*\otimes(e_1+e_2), \tag{5.2}$$
$$\begin{aligned}\widehat{\beta}_{\mathsf{N}}=\frac{4}{3}\Bigg(&\left[\frac{\sqrt{3}}{2}e_1^*+\frac{1}{2}e_2^*\right]\otimes\left[\frac{\sqrt{3}}{2}e_1^*+\frac{1}{2}e_2^*\right]\otimes\left[\frac{1}{2}e_1+\frac{\sqrt{3}}{2}e_2\right]\\&+\left[\frac{\sqrt{3}}{2}e_1^*-\frac{1}{2}e_2^*\right]\otimes\left[\frac{\sqrt{3}}{2}e_1^*-\frac{1}{2}e_2^*\right]\otimes\left[\frac{1}{2}e_1-\frac{\sqrt{3}}{2}e_2\right]-e_2^*\otimes e_2^*\otimes e_1\Bigg).\end{aligned} \tag{5.3}$$

5.1 Bilinear stability of complex multiplication algorithms

Recall from Sect. 2 that $\operatorname{rank}(\beta_{\mathbb{C}})=3=\overline{\operatorname{rank}}(\beta_{\mathbb{C}})$, i.e., both Gauss's algorithm and our new algorithm have optimal bilinear complexity whether in the exact or approximate sense. One may also show that $\beta_{\mathbb{C}}$ has nuclear norm [15, Lemma 6.1] is given by

$$\|\beta_{\mathbb{C}}\|_\nu=4.$$

The growth factor of the regular algorithm (5.1) attains this minimum value,

$$\begin{aligned}\gamma(\widehat{\beta}_{\mathsf{R}})&=\|e_1^*\|_*\|e_1^*\|_*\|e_1\|+\|-e_2^*\|_*\|e_2^*\|_*\|e_1\|+\|e_1^*\|_*\|e_2^*\|_*\|e_2\|\\&\quad+\|e_2^*\|_*\|e_1^*\|_*\|e_2\|\\&=4=\|\beta_{\mathbb{C}}\|_\nu,\end{aligned}$$

as does our new algorithm (5.3),

$$\begin{aligned}\gamma(\widehat{\beta}_{\mathsf{N}})=\frac{4}{3}\Bigg(&\left\|\frac{\sqrt{3}}{2}e_1^*+\frac{1}{2}e_2^*\right\|_*\left\|\frac{\sqrt{3}}{2}e_1^*+\frac{1}{2}e_2^*\right\|_*\left\|\frac{1}{2}e_1+\frac{\sqrt{3}}{2}e_2\right\|\\&+\left\|\frac{\sqrt{3}}{2}e_1^*-\frac{1}{2}e_2^*\right\|_*\left\|\frac{\sqrt{3}}{2}e_1^*-\frac{1}{2}e_2^*\right\|_*\left\|\frac{1}{2}e_1-\frac{\sqrt{3}}{2}e_2\right\|+\|e_2^*\|_*\|e_2^*\|_*\|e_1\|\Bigg)\\=4=\|\beta_{\mathbb{C}}\|_\nu,&\end{aligned}$$

but not Gauss's algorithm (5.2),

$$\begin{aligned}\gamma(\widehat{\beta}_{\mathsf{G}}) &= \|e_1^* + e_2^*\|_* \|e_1^* + e_2^*\|_* \|e_2\| + \|e_1^*\|_* \|e_1^*\|_* \|e_1 - e_2\| \\ &\quad + \|-e_2^*\|_* \|e_2^*\|_* \|e_1 + e_2\| \\ &= 2(1+\sqrt{2}) > \|\beta_{\mathbb{C}}\|_\nu.\end{aligned}$$

So Gauss's algorithm $\widehat{\beta}_{\mathsf{G}}$ is faster (by bilinear complexity) but less stable (by bilinear stability) than the regular algorithm. Our new algorithm $\widehat{\beta}_{\mathsf{N}}$ on the other hand is optimal in both measures, attaining both $\operatorname{rank}(\beta_{\mathbb{C}})$ and $\|\beta_{\mathbb{C}}\|_\nu$.

We stress that numerical stability is too complicated an issue to be completely covered by the simple framework of bilinear stability. For instance, from the perspective of cancellation errors, our new algorithm also suffers from the issue pointed out in [19, Section 23.2.4] for Gauss's algorithm. By choosing $z = w$ and $b = \sqrt{3}/a$, our algorithm (5.3) computes

$$\frac{1}{2}\left[\left(a+\frac{1}{a}\right)^2 + \left(a-\frac{1}{a}\right)^2 - \frac{8}{a^2}\right] + \frac{i\sqrt{3}}{2}\left[\left(a+\frac{1}{a}\right)^2 - \left(a-\frac{1}{a}\right)^2\right] =: x + iy.$$

There will be cancellation error in the computed real part $\widehat{x}$ when $|a|$ is small and likewise in the computed imaginary part $\widehat{y}$ when $|a|$ is large. Nevertheless, as discussed in [19, Section 23.2.4], the new algorithm (5.3) is still stable in the weaker sense of having acceptably small $|x - \widehat{x}|/|z|$ and $|y - \widehat{y}|/|z|$ even if $|x - \widehat{x}|/|x|$ or $|y - \widehat{y}|/|y|$ might be large.

5.2 Error analysis of new algorithm applied to matrices

While using Gauss's algorithm or our new algorithm for multiplying of complex *numbers* is a pointless overkill, they become useful when applied to the multiplication of complex *matrices*. Note that any complex matrices $A + iB, C + iD \in \mathbb{C}^{n\times n}$ may be multiplied via their real and imaginary parts $A, B, C, D \in \mathbb{R}^{n\times n}$:

$$(A+iB)(C+iD) = (AC - BD) + i[AD + BC], \tag{5.4}$$

allowing us to focus our attention on designing algorithms for real matrix products. In this regard, Gauss's algorithm applied in the form

$$(A+iB)(C+iD) = (AC - BD) + i[(A+B)(C+D) - AC - BD] \tag{5.5}$$

reduces the number of real matrix products from four to three at the expense of more matrix additions. This represents an enormous saving as matrix products are invariably much more expensive than matrix additions. Our new algorithm (1.5) likewise applies in the form

$$\begin{aligned}(A+iB)(C+iD)\\ &= \frac{1}{2}\left[\left(A+\frac{1}{\sqrt{3}}B\right)\left(C+\frac{1}{\sqrt{3}}D\right)+\left(A-\frac{1}{\sqrt{3}}B\right)\left(C-\frac{1}{\sqrt{3}}D\right)-\frac{8}{3}BD\right]\\ &\quad+\frac{i\sqrt{3}}{2}\left[\left(A+\frac{1}{\sqrt{3}}B\right)\left(C+\frac{1}{\sqrt{3}}D\right)-\left(A-\frac{1}{\sqrt{3}}B\right)\left(C-\frac{1}{\sqrt{3}}D\right)\right], \end{aligned}\tag{5.6}$$

trading expensive matrix products for inexpensive scalar multiplications and additions.

The following is an error analysis of (5.6), i.e., our new algorithm applied to complex matrix multiplication. We emulate a similar analysis for Gauss's algorithm in [17, 19], assuming in particular that the real matrix multiplications involved are performed using the conventional algorithm (as opposed to Strassen's or Winograd's). We remind the reader that conventional matrix multiplication has the simple error bound

$$|AB-\mathsf{fl}(AB)|\le n|A||B|\mathbf{u}+O(\mathbf{u}^2)\tag{5.7}$$

for $A, B\in\mathbb{R}^{n\times n}$.

Theorem 5.1 (Error analysis for our new algorithm) *Let $(A+iB)(C+iD)=F+iG$ with $F, G\in\mathbb{R}^{n\times n}$ and let $\widehat{F}_{\mathsf{N}}, \widehat{G}_{\mathsf{N}}$ be computed via (5.6) in floating point arithmetic satisfying (3.3). Then*

$$\begin{aligned}|F-\widehat{F}_{\mathsf{N}}| &\le (n+7)\left(|A|+\frac{1}{\sqrt{3}}|B|\right)\left(|C|+\frac{1}{\sqrt{3}}|D|\right)\mathbf{u}\\ &\quad+\left(\frac{4}{3}n+4\right)|B||D|\mathbf{u}+O(\mathbf{u}^2),\end{aligned}\tag{5.8}$$

$$|G-\widehat{G}_{\mathsf{N}}|\le\sqrt{3}(n+6)\left(|A|+\frac{1}{\sqrt{3}}|B|\right)\left(|C|+\frac{1}{\sqrt{3}}|D|\right)\mathbf{u}+O(\mathbf{u}^2),\tag{5.9}$$

where the inequality $\le$ and absolute value $|\cdot|$ both apply in a coordinatewise sense.

Proof Following [19], we use the same letter δ to denote the error incurred in each step of our algorithm. So, for example,

$$\mathsf{fl}(B/\sqrt{3})=B/\sqrt{3}+\delta B/\sqrt{3}.$$

In the following we will define matrices H_i and let $\widehat{H_i}$ be its computed value, $i=1,\dots,8$.

Let $H_1:=A+B/\sqrt{3}$. Then

$$\begin{aligned}\widehat{H_1}&=\mathsf{fl}(A+B/\sqrt{3}+\delta B/\sqrt{3})=(A+B/\sqrt{3}+\delta B/\sqrt{3})(1+\delta)\\ &=A+B/\sqrt{3}+\delta(A+2B/\sqrt{3})+O(\mathbf{u}^2)\\ &=H_1+2\Delta_1+O(\mathbf{u}^2),\qquad|\Delta_1|\le(|A|+|B|/\sqrt{3})\mathbf{u}.\end{aligned}$$

Similarly $H_2 := C + D/\sqrt{3}$ satisfies

$$\widehat{H}_2 = H_2 + 2\Delta_2 + O(\mathbf{u}^2), \qquad |\Delta_2| \le (|C| + |D|/\sqrt{3})\mathbf{u}.$$

Let $H_3 := (A + B/\sqrt{3})(C + D/\sqrt{3})$. By (5.7),

$$\widehat{H}_3 = (A + B/\sqrt{3} + 2\Delta_1)(C + D/\sqrt{3} + 2\Delta_2) + n\Delta_3 + O(\mathbf{u}^2) \tag{5.10}$$

where

$$\begin{aligned} |\Delta_3| &\le |(A + B/\sqrt{3} + 2\Delta_1)||(C + D/\sqrt{3} + 2\Delta_2)|\mathbf{u} \\ &\le (|A| + |B|/\sqrt{3} + 2|\Delta_1|)(|C| + |D|/\sqrt{3} + 2|\Delta_2|)\mathbf{u} \\ &\le (|A| + |B|/\sqrt{3} + 2\mathbf{u}(|A| + |B|/\sqrt{3}))(|C| + |D|/\sqrt{3} + 2\mathbf{u}(|C| + |D|/\sqrt{3}))\mathbf{u} \\ &\le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u} + O(\mathbf{u}^2). \end{aligned} \tag{5.11}$$

By (5.10) and (5.11),

$$\begin{aligned} \widehat{H}_3 &= (A + B/\sqrt{3})(C + D/\sqrt{3}) + 2\Delta_1(C + D/\sqrt{3}) \\ &\quad + 2(A + B/\sqrt{3})\Delta_2 + n\Delta_3 + O(\mathbf{u}^2) \\ &= H_3 + (n+4)\Delta_4 + O(\mathbf{u}^2) \end{aligned} \tag{5.12}$$

where

$$|\Delta_4| \le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u}.$$

Similarly $H_4 := (A - B/\sqrt{3})(C - D/\sqrt{3})$ satisfies

$$\widehat{H}_4 = H_4 + (n+4)\Delta_5 + O(\mathbf{u}^2) \tag{5.13}$$

where

$$|\Delta_5| \le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u}.$$

Let $H_5 := (A + B/\sqrt{3})(C + D/\sqrt{3}) + (A - B/\sqrt{3})(C - D/\sqrt{3})$. By (5.12) and (5.13),

$$\begin{aligned} \widehat{H}_5 &= [H_3 + (n+4)\Delta_4 + H_4 + (n+4)\Delta_5](1+\delta) + O(u^2) \\ &= H_5 + (2n+10)\Delta_6 + O(\mathbf{u}^2) \end{aligned} \tag{5.14}$$

where

$$|\Delta_6| \le \mathbf{u}(|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3}).$$

Let $H_6 := 8/3BD$. Then

$$\begin{aligned}\widehat{H}_6 &= \mathrm{fl}(8/3(BD + n\Delta_7)) + O(\mathbf{u}^2)\\ &= 8/3(BD + n\Delta_7)(1+\delta) + O(\mathbf{u}^2)\\ &= H_6 + 8/3(n+1)\Delta_8 + O(\mathbf{u}^2)\end{aligned} \tag{5.15}$$

where

$$|\Delta_7| \le |B||D|\mathbf{u}, \qquad |\Delta_8| \le |B||D|\mathbf{u}.$$

Let $H_7 := H_5 - H_6$. By (5.14) and (5.15),

$$\begin{aligned}\widehat{H}_7 &= [H_5 + (2n+10)\Delta_6 - H_6 - 8/3(n+1)\Delta_8](1+\delta) + O(\mathbf{u}^2)\\ &= H_7 + (2n+12)\Delta_9 + 8/3(n+2)\Delta_{10} + O(\mathbf{u}^2)\end{aligned}$$

where

$$|\Delta_9| \le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u}, \qquad |\Delta_{10}| \le |B||D|\mathbf{u}.$$

Then

$$\begin{aligned}\widehat{F}_{\mathsf{N}} &= (1+\delta)[H_7 + (2n+12)\Delta_9 + 8/3(n+2)\Delta_{10}]/2 + O(\mathbf{u}^2)\\ &= F + (n+7)\Delta_{11} + 4/3(n+3)\Delta_{12} + O(\mathbf{u}^2)\end{aligned}$$

where

$$|\Delta_{11}| \le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u}, \qquad |\Delta_{12}| \le |B||D|\mathbf{u},$$

and from which we obtain (5.8).

Let $H_8 := (A + B/\sqrt{3})(C + D/\sqrt{3}) - (A - B/\sqrt{3})(C - D/\sqrt{3})$. Similar to (5.14), we have

$$\widehat{H}_8 = H_8 - (2n+10)\Delta_{13} + O(\mathbf{u}^2)$$

where

$$|\Delta_{13}| \le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u}.$$

Then

$$\begin{aligned}\widehat{G}_{\mathsf{N}} &= \sqrt{3}/2[H_8 - (2n+10)\Delta_{13}](1+\delta) + O(\mathbf{u}^2)\\ &= G + \sqrt{3}(n+6)\Delta_{14} + O(\mathbf{u}^2)\end{aligned}$$

where

$$|\Delta_{14}| \le (|A| + |B|/\sqrt{3})(|C| + |D|/\sqrt{3})\mathbf{u},$$

from which we obtain (5.9). □

If we compute the matrices F, G in Theorem 5.1 using Gauss's algorithm (5.5) with floating point arithmetic and let the results be $\widehat{F}_{\mathsf{G}}$ and $\widehat{G}_{\mathsf{G}}$, then the corresponding error bounds [17, 19] are

$$\begin{aligned} |F - \widehat{F}_{\mathsf{G}}| &\le (n+1)(|A||C| + |B||D|)\mathbf{u} + O(\mathbf{u}^2), \\ |G - \widehat{G}_{\mathsf{G}}| &\le (n+4)[(|A|+|B|)(|C| + |D|) + |A||C|+|B||D|]\mathbf{u}+O(\mathbf{u}^2). \end{aligned} \tag{5.16}$$

When $n \to \infty$, we have $n + c \approx n$ for any constant c. Hence the errors in (5.8) and (5.9) are dominated by

$$\begin{aligned} |F - \widehat{F}_{\mathsf{N}}| &\sim n\left[|A||C| + \frac{5}{3}|B||D| + \frac{1}{\sqrt{3}}|A||D| + \frac{1}{\sqrt{3}}|B||C|\right]\mathbf{u}, \\ |G - \widehat{G}_{\mathsf{N}}| &\sim n\left[\sqrt{3}|A||C| + |B||C| + |A||D| + \frac{1}{\sqrt{3}}|B||D|\right]\mathbf{u}, \end{aligned}$$

whereas those in (5.16) are dominated by

$$\begin{aligned} |F - \widehat{F}_{\mathsf{G}}| &\sim n(|A||C| + |B||D|)\mathbf{u}, \\ |G - \widehat{G}_{\mathsf{G}}| &\sim n(2|A||C| + 2|B||D| + |A||D| + |B||C|)\mathbf{u}. \end{aligned}$$

For easy comparison suppose the magnitudes of the entries in A, B, C, D are all approximately θ, then these reduce to

$$\begin{aligned} |F - \widehat{F}_{\mathsf{N}}| &\sim 3.8n^2\theta^2, \qquad & |G - \widehat{G}_{\mathsf{N}}| &\sim 4.3n^2\theta^2, \\ |F - \widehat{F}_{\mathsf{G}}| &\sim 2n^2\theta^2, & |G - \widehat{G}_{\mathsf{G}}| &\sim 6n^2\theta^2. \end{aligned} \tag{5.17}$$

So Gauss's algorithm gives an imaginary part that is three times less accurate than its real part. Note the the imaginary part of Gauss's algorithm accounts for all its computational savings; the real part is just the regular algorithm. On the other hand, our algorithm balances the accuracy of both the real and imaginary parts by spreading out the computational savings across both parts.

To quantify this, we use the *max norm*. For a complex matrix $A + iB \in \mathbb{C}^{n\times n}$, this is

$$\|A + iB\|_{\max} := \max\{|a_{ij}|, |b_{ij}| : i, j = 1, \dots, n\}. \tag{5.18}$$

The max norm differs from the usual matrix ∞-norm given by maximum row sum used in [17, 19]. We favor the max norm as it is the strictest measure of numerical accuracy — a small max norm error implies that each entry is accurate as opposed to accurate on average.

If we denote the matrices resulting from Gauss's algorithm and our new algorithm by

$$\widehat{E}_{\mathsf{G}} := \widehat{F}_{\mathsf{G}} + i\widehat{G}_{\mathsf{G}}, \qquad \widehat{E}_{\mathsf{N}} := \widehat{F}_{\mathsf{N}} + i\widehat{G}_{\mathsf{N}}$$

respectively, we expect $\|E - \widehat{E}_{\mathsf{N}}\|_{\max}$ to be smaller than $\|E - \widehat{E}_{\mathsf{G}}\|_{\max}$. The extensive experiments in Sect. 6 will attest to this.

5.3 Derivation of our algorithm

It is perhaps instructive to include a description of how one may derive the algorithm in (1.5) by minimizing growth factor. Observe that Gauss's algorithm (1.1) includes the term $(a+b)(c+d)$, which adds 2 to its growth factor. We seek to reduce the growth factor by replacing it with $(a+rb)(c+rd)$ for some shrinkage $r \in (0,1)$, which leads to a family of algorithms parameterized by r:

$$\begin{aligned}(a+bi)(c+di) &= \frac{1}{2}[(a+rb)(c+rd) + (a-rb)(c-rd) - (2r^2+2)bd] \\ &\quad + \frac{i}{2r}[(a+rb)(c+rd) - (a-rb)(c-rd)].\end{aligned}$$

Let $g(r)$ denote the growth factor. A simple calculation shows that

$$g(r) = \frac{1}{r}(1+r^2)^{3/2} + r^2 + 1,$$

which has a minimum of 4 attained at $r = 1/\sqrt{3}$, giving us (1.5). Note that (1.5) is not unique; another algorithm with growth factor 4 is given by

$$\begin{aligned}(a+bi)(c+di) &= \frac{\sqrt{3}}{2}\left[\left(a+\frac{1}{\sqrt{3}}b\right)\left(\frac{1}{\sqrt{3}}c-d\right) + \left(a-\frac{1}{\sqrt{3}}b\right)\left(\frac{1}{\sqrt{3}}c+d\right)\right] \\ &\quad + \frac{i}{2}\left[\left(a-\frac{1}{\sqrt{3}}b\right)\left(\frac{1}{\sqrt{3}}c+d\right) - \left(a+\frac{1}{\sqrt{3}}b\right)\left(\frac{1}{\sqrt{3}}c-d\right) + \frac{8}{3}bc\right],\end{aligned}$$

which may be obtained from (1.5) by substituting $c = di$ and $d = -ci$.

6 Experiments for new complex matrix multiplication algorithm

The goal of this section is to provide numerical evidence to show that our new algorithm (5.6) for complex matrix multiplication is

- nearly as stable as the regular algorithm (5.4), and
- nearly as fast as Gauss's algorithm (5.5).

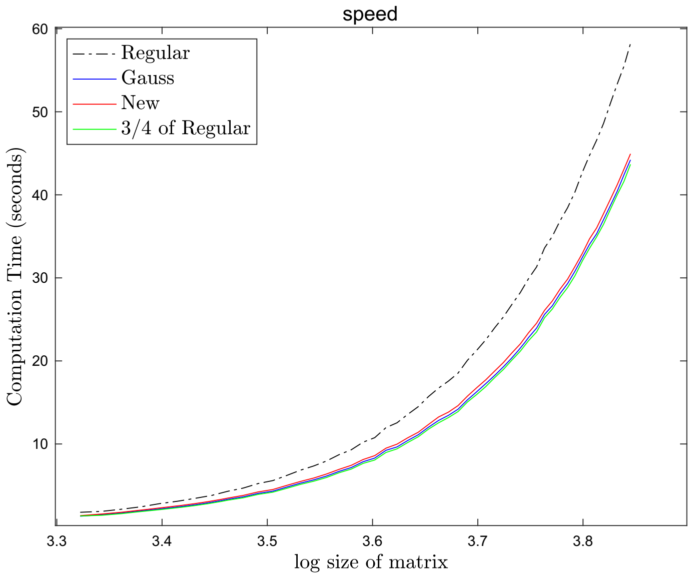


Fig. 2 Speed of the three algorithms for complex matrix multiplication

We begin with routine experiments comparing the three algorithms (5.4), (5.5), (5.6) on random matrices, and move on to three actual applications: matrix polynomial evaluations, unitary transformations, and the increasingly popular complex-valued neural networks. The results, we think, show that our new algorithm can be a realistic replacment for Gauss's algorithm in engineering applications.

6.1 Speed of the algorithms

We generate random $A + iB, C + iD \in \mathbb{C}^{n\times n}$ with entries of A, B, C, D drawn uniformly in $[-1, 1]$; the results with standard normal are similar and omitted. We increase n from 2100 to 7000 in steps of 100. The product $(A + iB)(C + iD)$ is computed numerically with the regular algorithm (5.4), Gauss's algorithm (5.5), and our new algorithm (5.6). For each n, we generate ten different matrices and record the average time taken for each algorithm and plot these in Fig. 2, with wall time (in seconds) for vertical axis and $\log_{10}(n)$ for horizontal axis. The time taken by MATLAB's internal function for complex matrix multiplication is virtually indistinguishable from that of the regular algorithm and therefore omitted.

Consistent with the predictions of bilinear complexity, our new algorithm has roughly the same computation time as Gauss's algorithm, at roughly 3/4 the time taken by the regular algorithm. We will perform more speed experiments in conjunction with our accuracy experiments in Sect. 6.2.

6.2 Accuracy of the algorithms

We generate random $A+iB$, $C+iD \in \mathbb{C}^{n\times n}$ with $n = 64, 128, 256$ and with condition numbers ranging from 174 to 3×10^{11}. We use the spectral condition number $\kappa_2(X)$, i.e., ratio of largest to smallest singular values of X, throughout this article. It is desirable to limit ourselves to matrices over Gaussian rationals, i.e., $\mathbb{Q} + \mathbb{Q}i$, as we will need to compute the exact values of their products later.

The way we generate such a matrix requires some elaboration. For an $X \in \mathbb{Z}^{n\times n}$ with a specified $\kappa_2(X) = \kappa \in \mathbb{Z}$. We form a diagonal $\Lambda \in \mathbb{R}^{n\times n}$ whose diagonal entries are 1 and κ toegether with $n-2$ other random integers between 1 and $\kappa - 1$. We then form $X = H\Lambda H^{\mathsf{T}}$ with a random Hadamard matrix $H \in \mathbb{Z}^{n\times n}$. If A and B are generated in this manner, then they are dense matrices (important as we do not want sparsity to unduly influence arithmetic costs) and $\kappa_2(A+iB) = \kappa_2(A) = \kappa_2(B) = \kappa$ as $(\kappa + \kappa i)/(1+i) = \kappa$.

We compute the exact value of $(A+iB)(C+iD)$ symbolically with MATLAB's symbolic toolbox. Given our relatively modest computational resources, this is the bottleneck for our experiments as this step becomes prohibitively expensive when $n > 256$. In generating the $n = 256$ plots in Fig. 3, this step alone took 40 h on our University's Research Computing Center servers.

For each pair of complex matrices $A+iB$ and $C+iD$, we compute their product $\widehat{E}$ using each of the three algorithms (5.4), (5.5), (5.6), and compare them against the exact result E via the max norm relative error

$$\frac{\|E - \widehat{E}\|_{\max}}{\|A+iB\|_{\max}\|C+iD\|_{\max}}.$$

As discussed in [17, 19], it is natural to measure error in matrix multiplication relative to the norms of the input matrices. We use the max norm in (5.18) to better capture entrywise accuracy.

The results are plotted in Fig. 3: speed plots have wall time in seconds on the vertical axes; accuracy plots have relative error on the vertical axes; all plots have $\log_{10}(\kappa)$ on the horizontal axes. We repeat each experiment ten times: every value on these plots comes from averaging across the results of ten pairs of random matrices with the same condition number.

Observations from Fig. 3: The accuracy of our new algorithm is much higher than that of Gauss's algorithm and only slightly worse than that of the regular algorithm. Gauss's algorithm also shows a great deal more fluctuation across varying condition numbers than either our new algorithm or the regular one. When it comes to speed, our algorithm is closer to that of Gauss's than the regular algorithm. These accuracy results attest to Theorem 5.1 and the discussions around (5.17).

The relative errors and wall times for MATLAB's internal function for complex matrix multiplication are virtually indistinguishable from those of the regular algorithm (that we implemented ourselves) and thus omitted. In the next three sections, we will compare the accuracy and speed of the three complex matrix multiplication algorithms in more realstic scenarios.

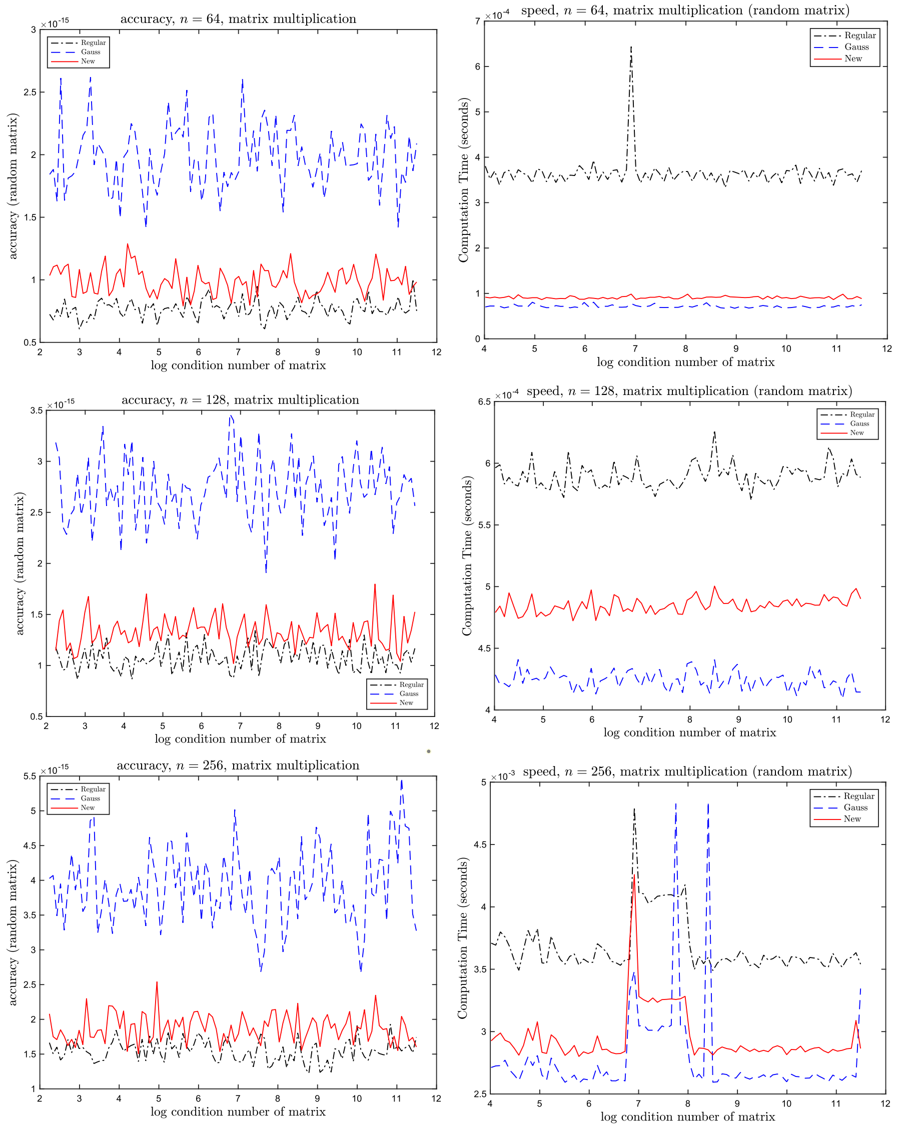


Fig. 3 Accuracy and speed of algorithms for complex matrix multiplication

6.3 Matrix polynomial evaluations

We evaluate a polynomial $p(x) = \sum_{k=0}^{d} a_k x^k$ with coefficients $a_0, \ldots, a_k \in \mathbb{R}$ at a $X \in \mathbb{C}^{n\times n}$. This is a problem that occurs in many tasks involving matrix functions [19, 20]. We limit ourselves to real coefficients as this is by far most common scenario [20]; but the complex coefficients case simply reduces to evaluating two real polynomials

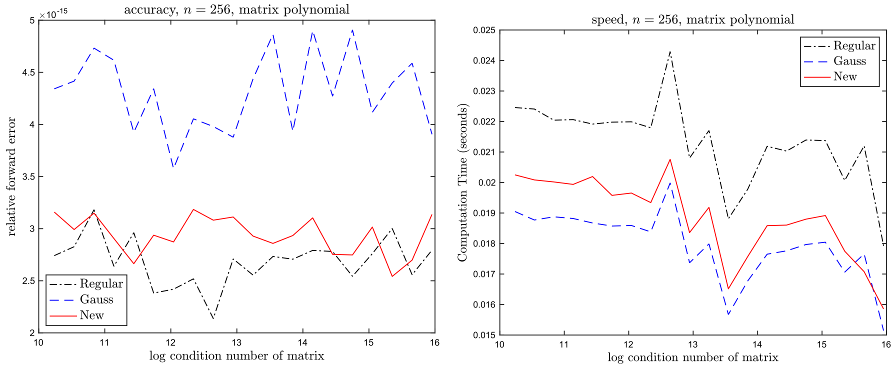


Fig. 4 The three algorithms applied to matrix polynomial evaluations

$\Re p(x)$ and $\Im p(x)$. The celebrated Horner's rule [20, Algorithm 4.3], as shown in Algorithm 1, reduces the problem to one of repeated matrix multiplications.

Algorithm 1 Compute $p(X)$ via Horner's rule

```
Input $a_0, a_1, \dots, a_d \in \mathbb{R}$, $X \in \mathbb{C}^{n\times n}$
Output $a_0 I + a_1 X + \cdots + a_d X^d$
1: $P = X$;
2: $S = a_0 I + a_1 X$;
3: for $k = 2 : d$ do
4:     $P = PX$;
5:     $S = S + a_k P$;
6: end for
7: return $S$;
```

We generate random matrices $X \in \mathbb{C}^{256\times 256}$ with condition numbers from 2^{34} to 2^{53} as described in Sect. 6.2. We set $d = 5$ and choose random $b_0, \dots, b_5 \in (0, 1)$ uniformly. We then evaluate $p(X)$ using Algorithm 1, with Step 4 computed via (5.4), (5.5), and (5.6).

We measure accuracy in terms of the max norm relative forward error

$$\frac{\|p(X) - \widehat{p}(X)\|_{\max}}{\|p(X)\|_{\max}},$$

using MATLAB symbolic toolbox for the exact value of $p(X)$. The results presented in Fig. 4 again show that our new algorithm is nearly as stable as the regular algorithm and nearly as fast as Gauss's algorithm. While our accuracy tests are again limited by our capacity for symbolic computation ($n = 256$ is fine, $n = 512$ is beyond reach), our speed tests can go far beyond (to around $n = 4096$), and they show a profile much like Fig. 2.

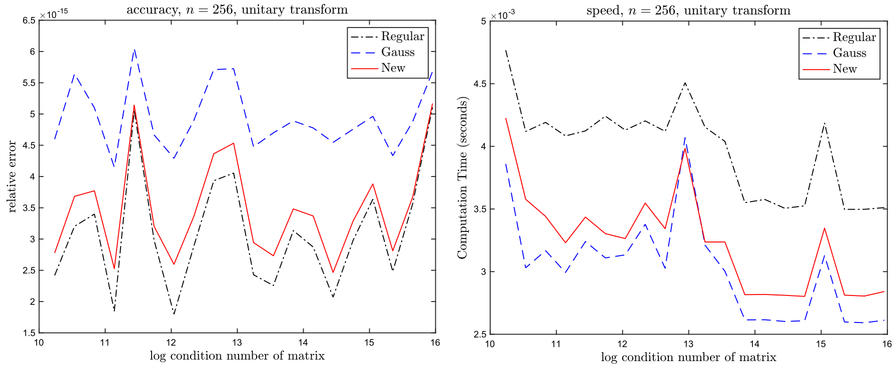


Fig. 5 The three algorithms applied to unitary transforms

6.4 Unitary transforms

Given a unitary matrix $U \in \mathbb{C}^{n\times n}$ and a complex matrix $X \in \mathbb{C}^{n\times n}$, it may come as a surprise to the reader that unless U happens to be some special transforms like FFT, DCT, DWT, etc, or has already been factored into a product of Householder or Givens matrices, there is no known special algorithm for forming UX that would take advantage of the unitarity of U. Nevertheless, such unitary matrices with no additional special structure are not uncommon. For instance, the matrix U could come from polar decompositions or matrix sign functions [16, 18, 22], and computed via iterative methods [16, 18, 22] and thus not in Householder- or Givens-factored form. Here we will explore the use of algorithms (5.4), (5.5), (5.6) for unitary transforms $X \mapsto UX$.

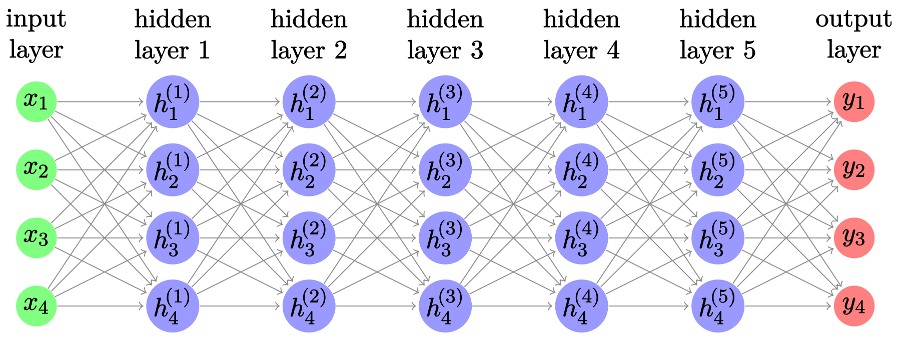
We generate the unitary matrix $U \in \mathbb{C}^{256\times 256}$ by QR factoring complex random matrices with entries in $\mathcal{U}[0, 1]+\mathcal{U}[0, 1]i$. Note that a unitary matrix is always perfectly conditioned. The matrix $X \in \mathbb{C}^{256\times 256}$ is generated randomly with condition numbers from 2^{34} to 2^{53} as in Sect. 6.3. We compute the exact value $E := UX$ symbolically as before and measure the accuracy of our computed value $\widehat{E}$ by

$$\frac{\|E - \widehat{E}\|_{\max}}{\|U\|_{\max}\|X\|_{\max}}.$$

The results, presented in Fig. 5, allow us to draw the same conclusion as in the Sect. 6.3. Further speed tests up to $n = 4096$ again show a profile much like Fig. 2.

6.5 Complex-valued neural networks

A complex-valued neural networks is simply a neural network with complex-valued weights and is activated by a complex function. It has become increasingly important and is widely used in signal processing and computer vision [1, 3, 9, 31, 36, 39]. For



input layer
hidden layer 1
hidden layer 2
hidden layer 3
hidden layer 4
hidden layer 5
output layer
x_1
x_2
x_3
x_4
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
$h_1^{(3)}$ $h_2^{(3)}$ $h_3^{(3)}$ $h_4^{(3)}$
$h_1^{(4)}$ $h_2^{(4)}$ $h_3^{(4)}$ $h_4^{(4)}$
$h_1^{(5)}$ $h_2^{(5)}$ $h_3^{(5)}$ $h_4^{(5)}$
y_1
y_2
y_3
y_4

Fig. 6 A constant width neural network with input dimension $n = 4$ and depth $d = 6$. The arrows between adjacent layers are weighted with values in the weight matrices. $h^{(k)} \in \mathbb{R}^n$ denotes the output of the kth layer

simplicity, we consider a d-layer constant width version $f : \mathbb{C}^n \to \mathbb{C}^n$ given by

$$f(W_1, \ldots, W_d, \sigma)(x) := W_d\sigma(W_{d-1}\sigma(\cdots W_2\sigma(W_1x)\cdots)),$$

with weight matrices $W_1, \ldots, W_d \in \mathbb{C}^{n\times n}$ and activation function $\sigma : \mathbb{C} \to \mathbb{C}$ applied coordinatewise on $\mathbb{C}^n$, as depicted in Fig. 6.

Complex matrix multiplications are indispensable when we *train* (i.e., fit with data in order to determine the weights $W_1, \ldots, W_d$) such a neural network through back-propagation, or when we *evaluate* it on multiple inputs $x_1, \ldots, x_m \in \mathbb{C}^n$ to make new predictions. Here we will compare the performance of the three algorithms (5.4), (5.5), (5.6) for the latter task as it allows for easier control of the condition numbers of $W_1, \ldots, W_d$.

For concreteness, we choose a depth of $d = 6$ and use the complex ReLU activation [3, 36]

$$\sigma(a + bi) := \max(a, 0) + \max(b, 0)i.$$

We generate random weight matrices $W_1, \ldots, W_6 \in \mathbb{C}^{n\times n}$ with $n = 64$ and 128, and with condition numbers ranging from 2^{34} to 2^{53}. We also generate random inputs $X = [x_1, \ldots, x_m] \in \mathbb{C}^{n\times m}$ with entries drawn from $\mathcal{U}[-\frac{1}{2}, \frac{1}{2}] + \mathcal{U}[-\frac{1}{2}, \frac{1}{2}]i$, and with $(m, n) = (25, 64)$ or $(50, 128)$. The task is then to evaluate

$$E := f(W_1, \ldots, W_d, \sigma)(X) := W_d\sigma(W_{d-1}\sigma(\cdots W_2\sigma(W_1X)\cdots)).$$

Again we compute its exact value E symbolically, apply the three algorithms to obtain $\widehat{E}$ numerically, and measure accuracy in terms of the relative forward error

$$\frac{\|E - \widehat{E}\|_{\max}}{\|E\|_{\max}}.$$

The results, shown in Fig. 7, are fully consistent with those in Sects. 6.3 and 6.4.

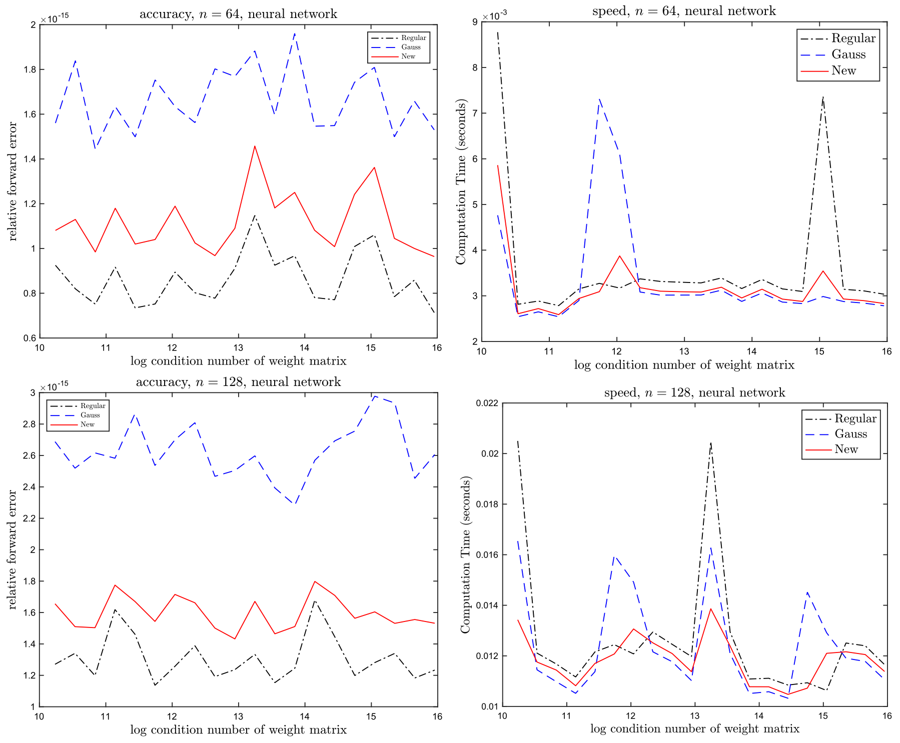


Fig. 7 The three algorithms applied to 6-layer complex neural networks with complex ReLU activation and widths 64 and 128

7 Conclusion

The notion of bilinear complexity started by Strassen has been a great motivator for more than five decades of exciting developments in numerical linear algebra. Its success illustrates the adage that "less is more". Bilinear complexity does not capture every operation that underlies the speed of an algorithm; but by focusing on a single operation (variable multiplications) and disregarding the rest (e.g., scalar multiplications, additions), it allows speed to be measured by the number of terms in a decomposition of a 3-tensor and the fastest algorithm to be given by a rank decomposition. This opens a door to other areas of mathematics like algebraic geometry where such decompositions are studied independent of their computational relevance.

We hope the notion of bilinear stability proposed in this article would do for the study of numerical stability what bilinear complexity did for the study of time complexity. By focusing on a single factor (growth) and disregarding other factors (e.g., cancellation errors) that play a role in numerical stability, it allows stability to be measured by the growth factor in a decomposition of a 3-tensor and the stablest algorithm to be given by a nuclear decomposition. Just as tensor rank connects to algebraic geometry, tensor nuclear norm connects to functional analysis [10, 12, 30]; thus bilinear stability could potentially open a door to this rich area of mathematics.

A very recent development in bilinear complexity is the automated discovery of fast algorithms using deep reinforcement learning. In [14], *AlphaTensor* found more than 14,000 inequivalent 49-term decompositions for 4×4 matrix product. This is impressive. But when one has that many different algorithms the question becomes which one to pick? From the perspective of numerical linear algebra, numerical stability would be the most natural secondary criteria. Since the 14,000 algorithms are all given in the form of 49-term decompositions, their growth factors are trivial to calculate and all one needs to do is to pick the decomposition with the smallest growth factor.

Acknowledgements The authors would like to thank Nick Higham and Ke Ye for helpful discussions, the two anonymous reviewers for their very pertinent suggestions, and the University of Chicago's Research Computing Center for its computing resources and services. ZD acknowledges the support of DARPA HR00112190040 and NSF ECCF 2216912. LHL acknowledges the support of DARPA HR00112190040, NSF DMS 1854831, and a Vannevar Bush Faculty Fellowship ONR N000142312863.

References

1. Aizenberg, I.: Complex-Valued Neural Networks with Multi-Valued Neurons. Studies in Computational Intelligence, vol. 353. Springer, Berlin (2011)
2. Ballard, G., Benson, A.R., Druinsky, A., Lipshitz, B., Schwartz, O.: Improving the numerical stability of fast matrix multiplication. SIAM J. Matrix Anal. Appl. **37**(4), 1382–1418 (2016)
3. Bassey, J., Qian, L., Li, X.: A survey of complex-valued neural networks. arXiv:2101.12249 (2021)
4. Bini, D., Lotti, G.: Stability of fast algorithms for matrix multiplication. Numer. Math. **36**(1), 63–72 (1980)
5. Bini, D., Lotti, G., Romani, F.: Approximate solutions for the bilinear form computational problem. SIAM J. Comput. **9**(4), 692–697 (1980)
6. Borodin, A., Munro, I.: The Computational Complexity of Algebraic and Numeric Problems. Elsevier Computer Science Library: Theory of Computation Series, No. 1. American Elsevier Publishing Co., Inc., New York-London-Amsterdam (1975)
7. Brent, R.P.: Algorithms for matrix multiplication. March 1970. Report Stan-CS-70-157, Stanford University
8. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. Grundlehren der Mathematischen Wissenschaften, vol. 315. Springer, Berlin (1997)
9. Dash, B.N., Khare, N.: Deep complex neural network applications in remote sensing: an introductory review. In: Ranney, K.I., Raynal, A.M. (eds.) Radar Sensor Technology XXV. 11742, pp. 34–44. International Society for Optics and Photonics, SPIE, Bellingham (2021)
10. Defant, A., Floret, K.: Tensor Norms and Operator Ideals. North-Holland Mathematics Studies, vol. 176. North-Holland Publishing Co., Amsterdam (1993)
11. Derksen, H.: On the nuclear norm and the singular value decomposition of tensors. Found. Comput. Math. **16**(3), 779–811 (2016)
12. Diestel, J., Fourie, J.H., Swart, J.: The Metric Theory of Tensor Products. American Mathematical Society, Providence (2008)
13. Fam, A.T.: Efficient complex matrix multiplication. IEEE Trans. Comput. **37**(7), 877–879 (1988)
14. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, F.J.R., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., Kohli, P.: Discovering faster matrix multiplication algorithms with reinforcement learning. Nature **610**, 47–53 (2022)
15. Friedland, S., Lim, L.-H.: Nuclear norm of higher-order tensors. Math. Comp. **87**(311), 1255–1281 (2018)
16. Higham, N.J.: Computing the polar decomposition–with applications. SIAM J. Sci. Stat. Comput. **7**(4), 1160–1174 (1986)

17. Higham, N.J.: Stability of a method for multiplying complex matrices with three real matrix multiplications. SIAM J. Matrix Anal. Appl. **13**(3), 681–687 (1992)
18. Higham, N.J.: The matrix sign decomposition and its relation to the polar decomposition. In: Proceedings of the 3rd ILAS Conference (Pensacola, FL, 1993) volume 212/213, pp. 3–20 (1994)
19. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2002)
20. Higham, N.J.: Functions of Matrices. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (2008)
21. Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Dokl. Akad. Nauk SSSR **14**(145), 293–294 (1962)
22. Kenney, C., Laub, A.J.: On scaling Newton's method for polar decomposition and the matrix sign function. SIAM J. Matrix Anal. Appl. **13**(3), 698–706 (1992)
23. Kljuev, V.V., Kokovkin-Ščerbak, N.I.: On the minimization of the number of arithmetic operations for solving linear algebraic systems of equations. Ž. Vyčisl. Mat i Mat. Fiz. **5**, 21–33 (1965)
24. Knuth, D.E.: The Art of Computer Programming, vol. 2, 3rd edn. Addison-Wesley, Reading (1998)
25. Landsberg, J.M.: The border rank of the multiplication of 2×2 matrices is seven. J. Am. Math. Soc. **19**(2), 447–459 (2006)
26. Landsberg, J.M.: Geometry and Complexity Theory Cambridge. Studies in Advanced Mathematics, vol. 169. Cambridge University Press, Cambridge (2017)
27. Lim, L.-H.: Tensors in computations. Acta Numer. **30**, 555–764 (2021)
28. Moore, C., Mertens, S.: The Nature of Computation. Oxford University Press, Oxford (2011)
29. Rudich, S.: Complexity theory: from Gödel to Feynman. In: *Computational Complexity Theory*, volume 10 of *IAS/Park City Math. Ser.*, pp. 5–87. Amer. Math. Soc., Providence, RI (2004)
30. Ryan, R.A.: Introduction to Tensor Products of Banach Spaces. Springer Monographs in Mathematics. Springer, London (2002)
31. Scardapane, S., Van Vaerenbergh, S., Hussain, A., Uncini, A.: Complex-valued neural networks with nonparametric activation functions. IEEE Trans. Emerg. Topics Comput. **4**(2), 140–150 (2018)
32. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**, 354–356 (1969)
33. Strassen, V.: Vermeidung von Divisionen. J. Reine Angew. Math. **264**, 184–202 (1973)
34. Strassen, V.: Relative bilinear complexity and matrix multiplication. J. Reine Angew. Math. **375**(376), 406–443 (1987)
35. Strassen, V.: Algebraic complexity theory. In: Handbook of Theoretical Computer Science, Vol. A. Elsevier, Amsterdam, pp. 633–672 (1990)
36. Trabelsi, C., Bilaniuk, O., Zhang, Y., Serdyuk, D., Subramanian, S., Santos, J.F., Mehri, S., Rostamzadeh, N., Bengio, Y., Pal, C.J.: Deep complex networks. In: International Conference on Learning Representations (2018)
37. Winograd, S.: On multiplication of 2×2 matrices. Linear Algebra Appl. **4**, 381–388 (1971)
38. Ye, K., Lim, L.-H.: Fast structured matrix computations: tensor rank and Cohn-Umans method. Found. Comput. Math. **18**(1), 45–95 (2018)
39. Zhang, H., Gu, M., Jiang, X., Thompson, J., Cai, H., Paesani, S., Santagati, R., Laing, A., Zhang, Y., Yung, M., et al.: An optical neural chip for implementing complex-valued neural network. Nat. Commun. **12**(1), 1–11 (2021)